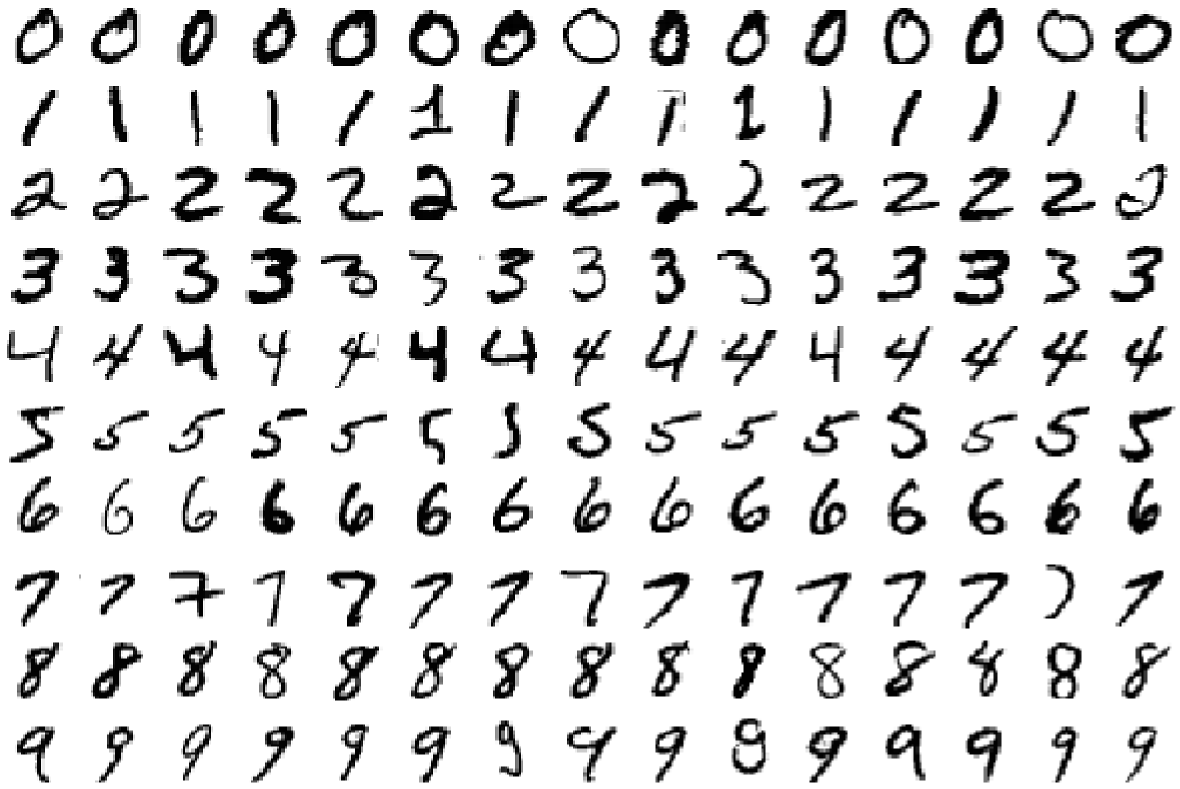
NEURAL NETWORKS AND PYTHON TUTORIALS PART 2

SETH HUANG

ANNOUNCEMENT

1. The first assignment is up
2. The related texts have been uploaded for this week
3. If you are auditing, you will still have to complete assignments and do the projects

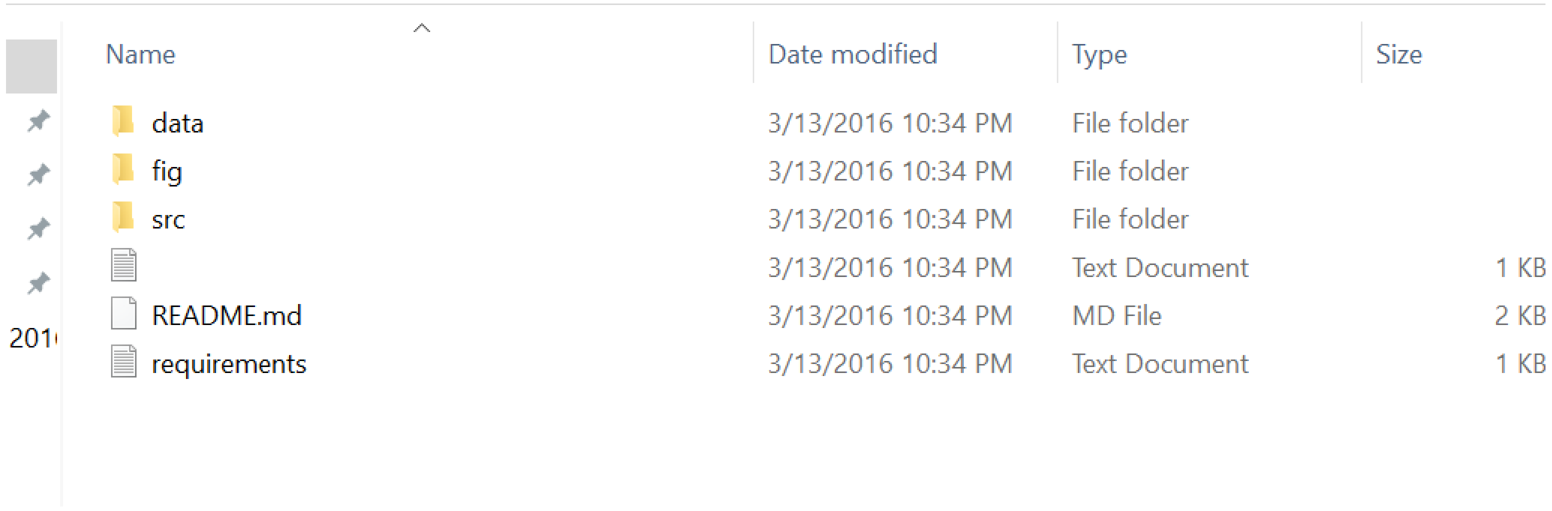
MNIST



http://yann.lecun.com/exdb/mnist/

DOWNLOAD CODES AND DATA HERE:

- HTTPS://GITHUB.COM/MNIELSEN/NEURAL-NETWORKS-AND-DEEP-LEARNING/ARCHIVE/MASTER.ZIP



Name	Date modified	Type	Size
data	3/13/2016 10:34 PM	File folder	
fig	3/13/2016 10:34 PM	File folder	
src	3/13/2016 10:34 PM	File folder	
	3/13/2016 10:34 PM	Text Document	1 KB
README.md	3/13/2016 10:34 PM	MD File	2 KB
requirements	3/13/2016 10:34 PM	Text Document	1 KB

MNIST DATA

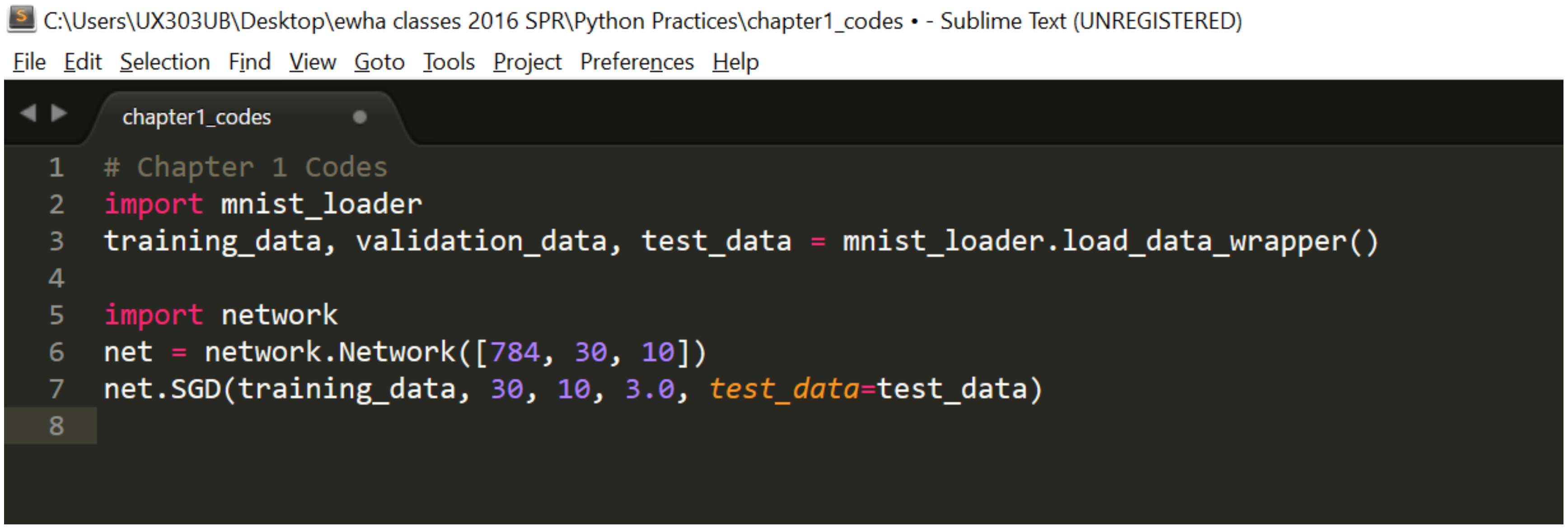
70000 28 x 28 images:

Split into:

1. 50000 training set
2. 10000 validation set
3. 10000 test set

MNIST DATA – ONE LAYER TRAINING

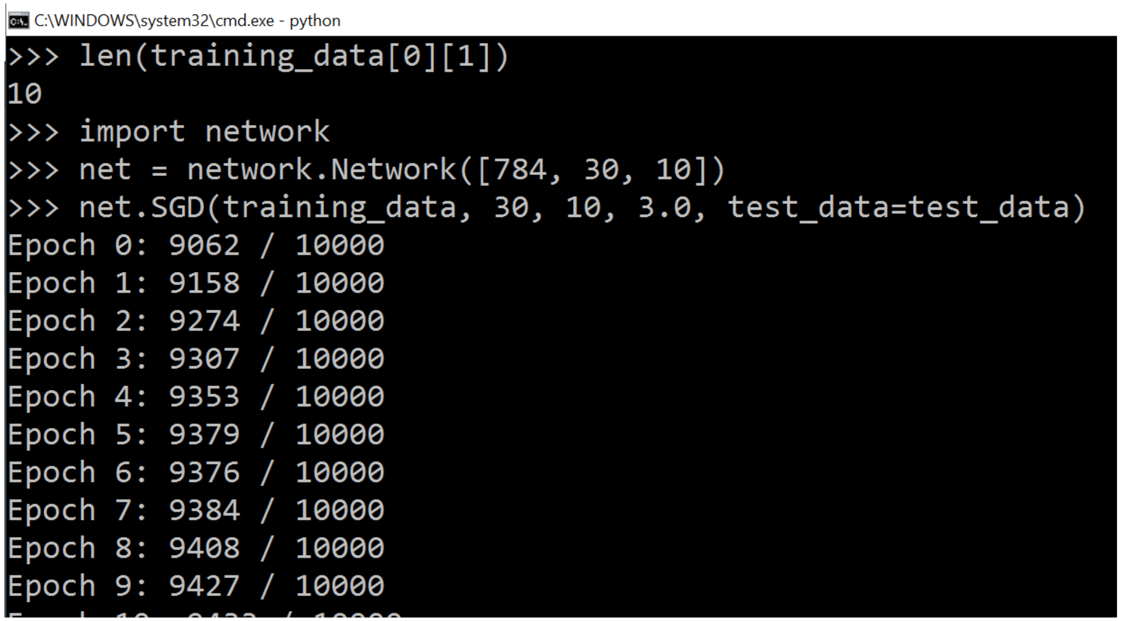
network.Network([784, 30, 10])
⇒ 784 dimensional input (28 x 28)
⇒ 30 neurons (in the hidden layer)
⇒ 10 outputs (0 – 9)


C:\Users\UX303UB\Desktop\ewha classes 2016 SPR\Python Practices\chapter1_codes • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
chapter1_codes

```python
# Chapter 1 Codes
import mnist_loader
training_data, validation_data, test_data = mnist_loader.load_data_wrapper()

import network
net = network.Network([784, 30, 10])
net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
```

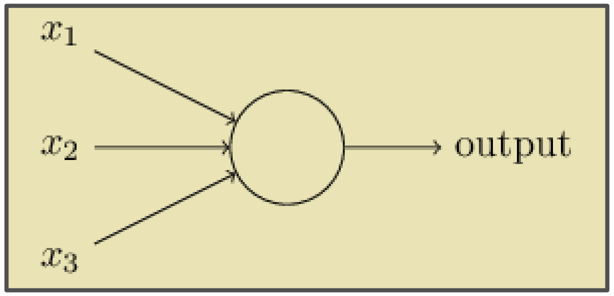
MNIST DATA – ONE LAYER TRAINING

SEE IF YOU CAN GET THIS TO WORK



```
>>> len(training_data[0][1])
10
>>> import network
>>> net = network.Network([784, 30, 10])
>>> net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
Epoch 0: 9062 / 10000
Epoch 1: 9158 / 10000
Epoch 2: 9274 / 10000
Epoch 3: 9307 / 10000
Epoch 4: 9353 / 10000
Epoch 5: 9379 / 10000
Epoch 6: 9376 / 10000
Epoch 7: 9384 / 10000
Epoch 8: 9408 / 10000
Epoch 9: 9427 / 10000
```

NOW TAKE LET'S DISCUSS NEURAL NETS IN DEPTH AS ENVISIONED BY FRANK ROSENBLATT

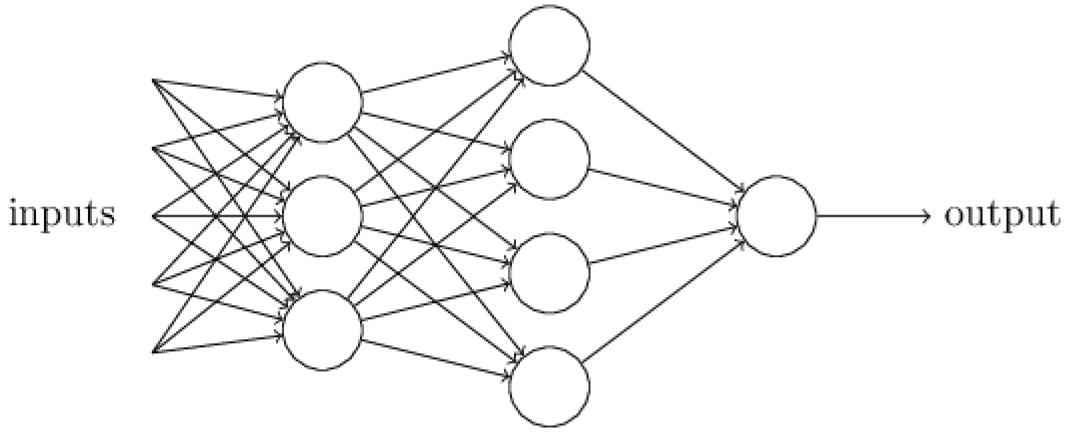


x1, x2, x3 are the inputs
w1, w2, w3 are the weights
0, 1 binary output
The neuron is a SIGNMOID

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{1}$$

THIS IS THE BASIC NEURAL NET

You can use the above model to answer various questions. Ex: if the threshold is greater than 2, then the weather is great, and if the threshold is lower than 2, everyone should stay home.



As discussed last week, there are several layers.

REWRITING THE CONDITIONS IN DOT PRODUCTS OF WEIGHT AND INPUT ~ (W, X)

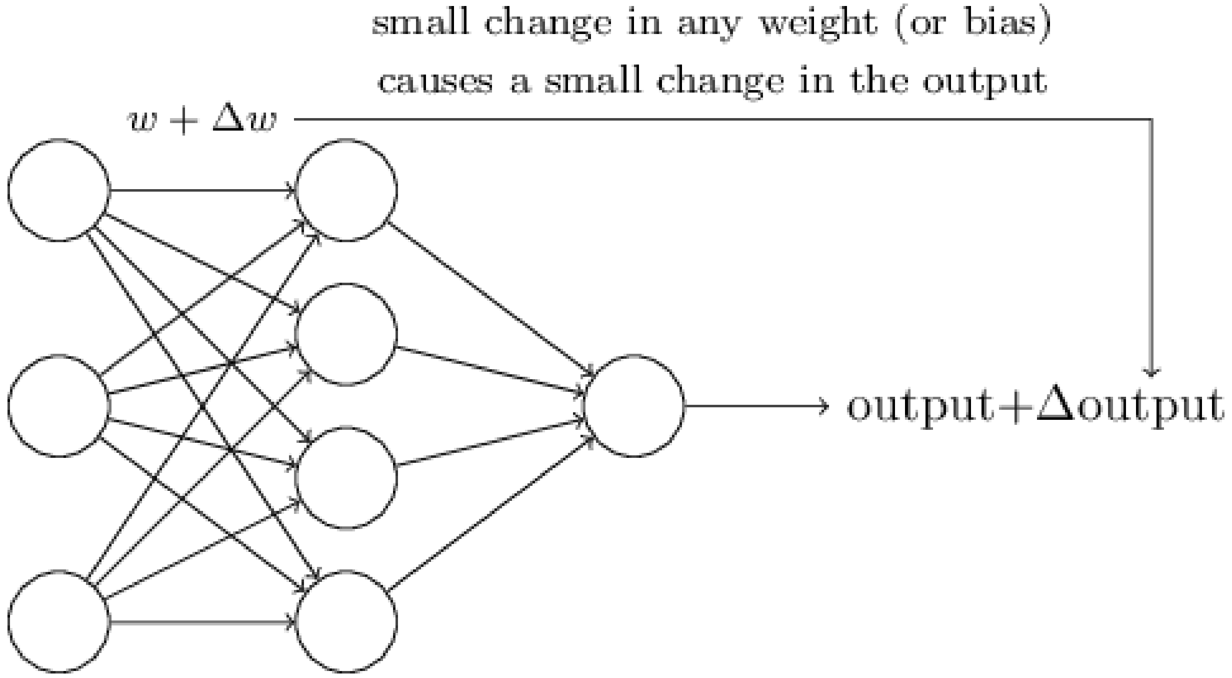
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{2}$$

Notice that b is just threshold moved to the other side, and it is known as “bias” from here on.

w and x are now weight and input vectors (you should know what this is)

SIGMOID NEURONS

Suppose we are training the mnist dataset, where the inputs are raw pixels.


small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

WHY ARE WE USING SIGMOID NEURONS OR OTHER “ACTIVATION FUNCTIONS?”

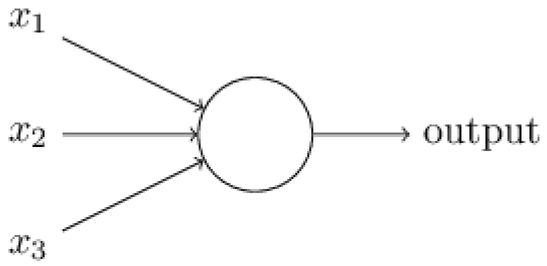
Suppose the network was mistakenly classifying an image as an "8" when it should be a "9". We could figure out how to make a small change in the weights and biases so the network gets a little closer to classifying the image as a "9"

BUT a small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip, say from 0 to 1.

$\Rightarrow$ We want the changes in weights and biases to HAVE SMALL EFFECTS IN THE OUTPUTS!

$\Rightarrow$ **THIS CONCEPT IS OFTEN NOT EXPLAINED WELL, BUT IF YOU THINK ABOUT THE PROBLEM THIS WAY, YOU UNDERSTAND ANY ACTIVITATION FUNCTION CAN POTENTIALLY BE USED.**

SIGMOID NEURONS



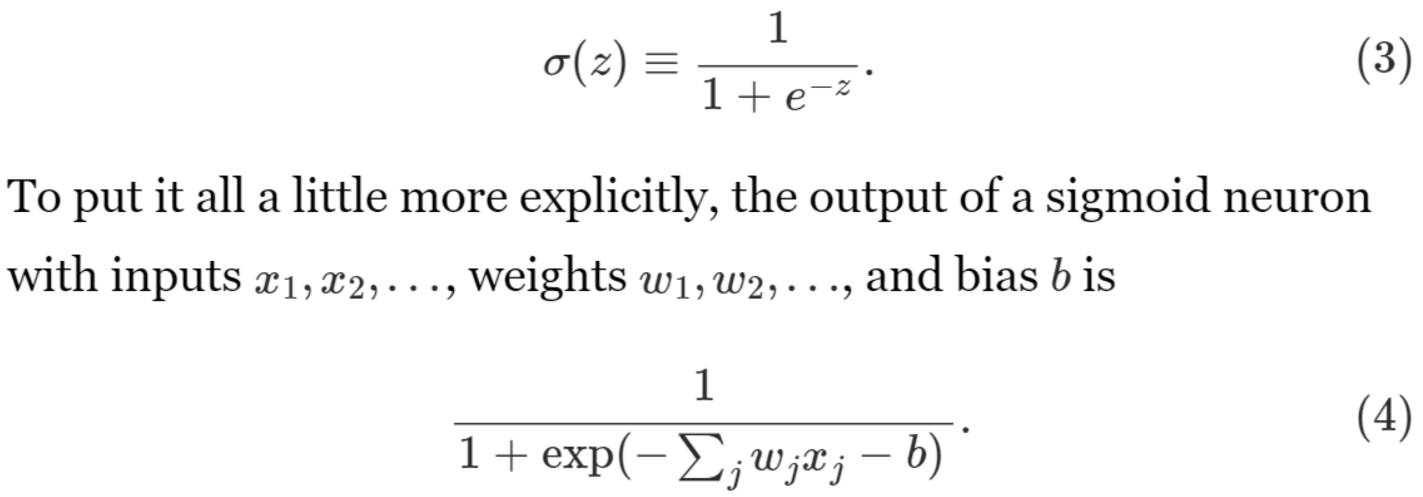
So, the input can take on any values. For example, greyscale can lie between 0 to 255 (black to white). Instead of having outputs like this,

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \qquad (2)$$

We will have $$\sigma(w \cdot x + b)$$

And the sigma is “sigmoid function.”

SIGMOID NEURONS



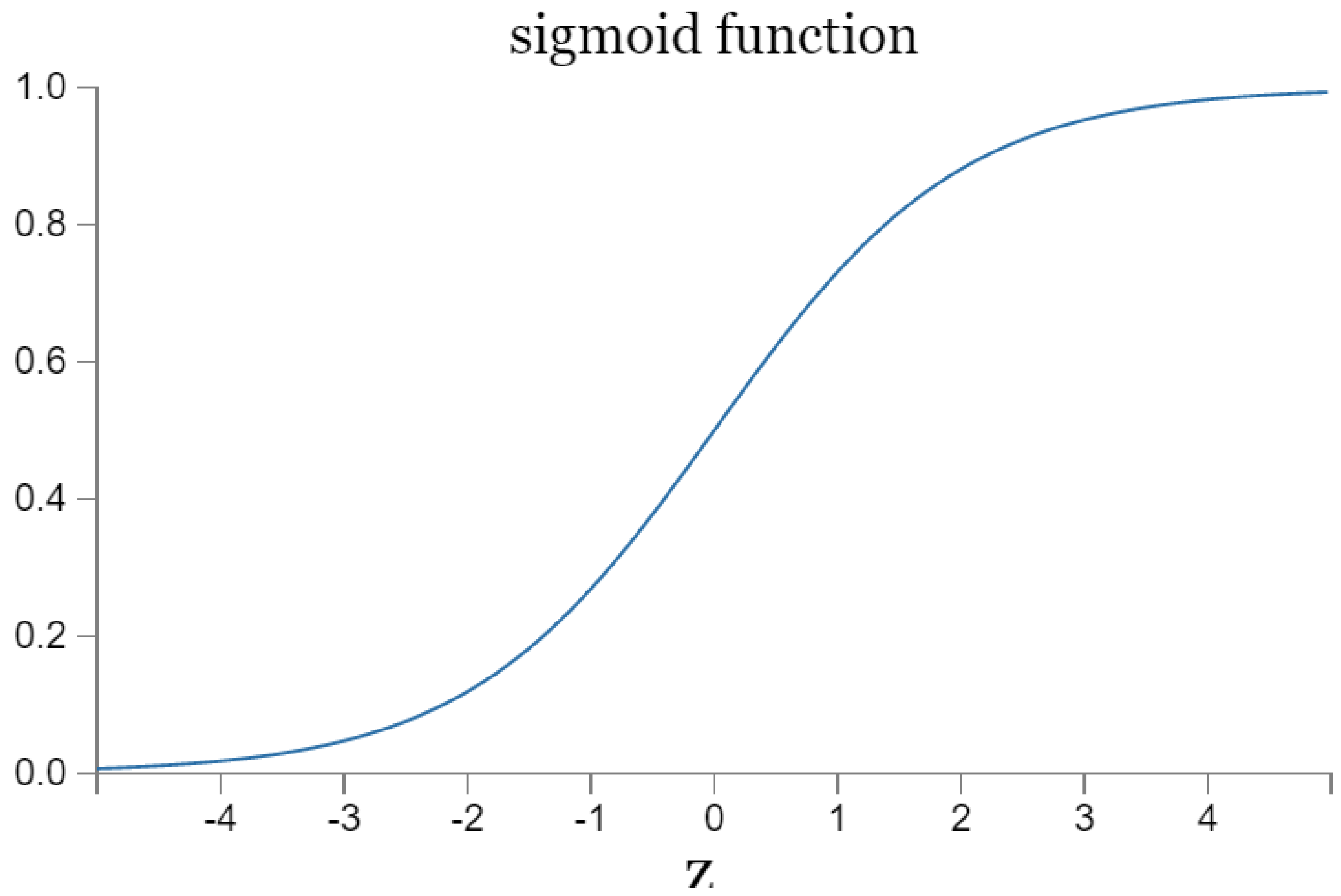
$$\sigma(z) \equiv \frac{1}{1+e^{-z}}. \qquad (3)$$

To put it all a little more explicitly, the output of a sigmoid neuron with inputs $x_1, x_2, \ldots$, weights $w_1, w_2, \ldots$, and bias b is

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}. \qquad (4)$$

What happens if bias or weight gets very large or very small?

SIGMOID NEURONS



SIGMOID NEURONS

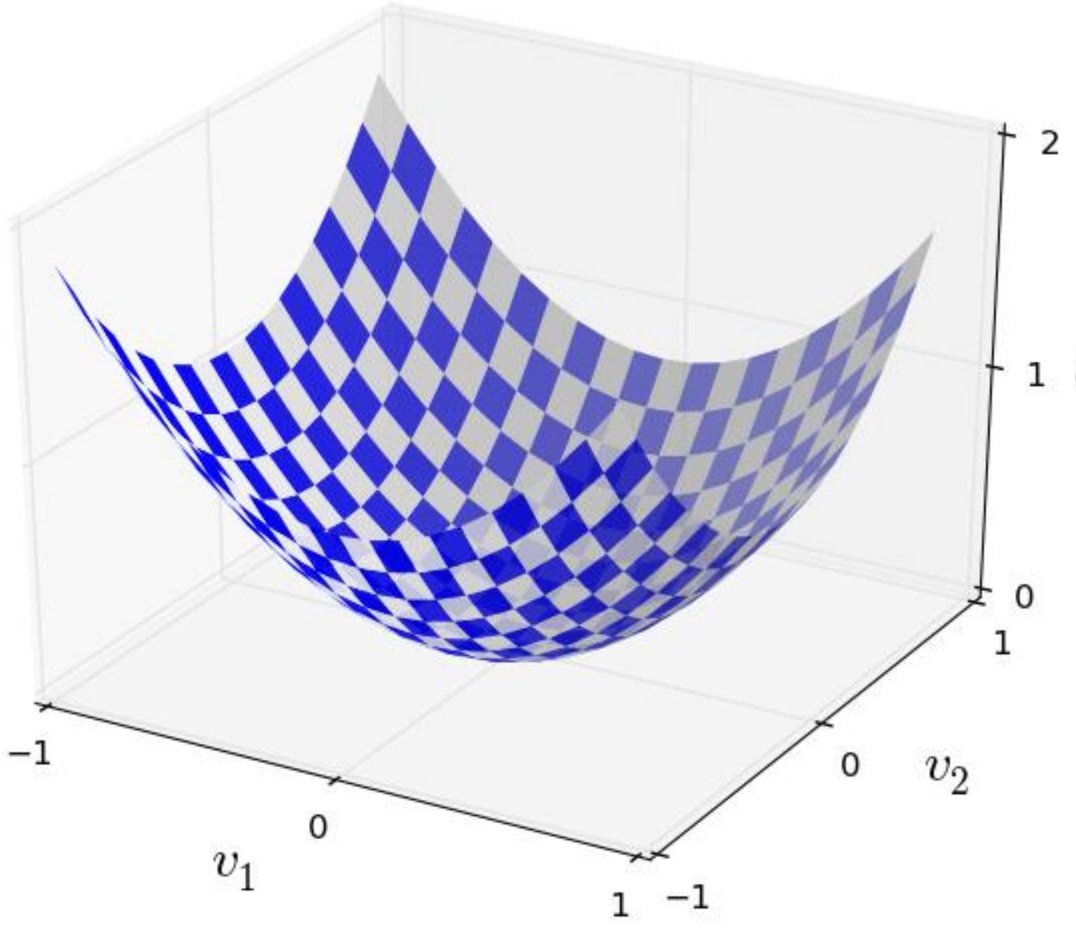
The smoothness of the function means small changes in weights and biases will result in small changes in the output.

$$\Delta\text{output} \approx \sum_j \frac{\partial\,\text{output}}{\partial w_j}\Delta w_j + \frac{\partial\,\text{output}}{\partial b}\Delta b, \tag{5}$$

Pay attention to the partial derivatives. What are they?

Also note that the changing output is a LINEAR function.

LEARNING WITH GRADIENT DESCENT



THE COST FUNCTION

Desired (perfect) output: $y = y(x)$

$$y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T$$

The cost function is thus:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \| y(x) - a \|^2.$$

a is the output given by the model, which is a function of x, w, and b, the input, weights and biases respectively.

$\|v\|$ Is just the length function for a vector v.

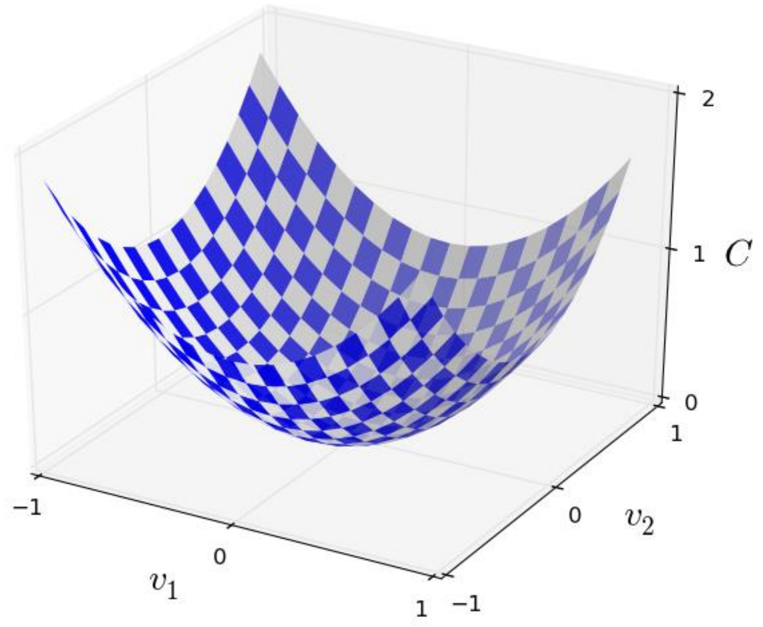
THE COST FUNCTION

First, the cost function is non-negative.

$$C(w, b) \equiv \frac{1}{2n} \sum_x \| y(x) - a \|^2$$

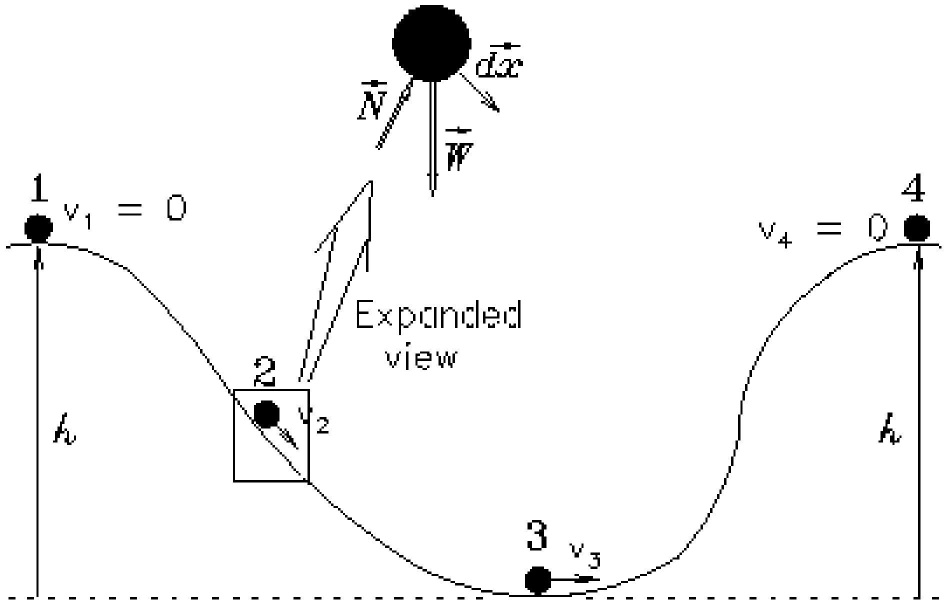
This cost function can be extended easily to incorporate different techniques such as cross-entropy function, which we will learn later on.

THE GRADIENT DESCENT



What we are doing is to try and find the global minima. Note that THE REAL COST FUNCTION IS NOT PRETTY LIKE THIS.

THE GRADIENT DESCENT CONCEPT



We pick a random starting point for a ball, and suppose the ball calculates the derivatives of the landscape. By studying these derivatives, we can understand what the landscape looks like.

THE GRADIENT DESCENT CONCEPT

Assuming v is direction the ball can move.

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2.$$

We are simply choosing the changing v (the directions) to make changing C negative. Thus, the gradient vector of C (the rate of change for C with respect to changing) is:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T.$$

∇ denotes the GRADIENT of a function.

THE GRADIENT DESCENT CONCEPT

This means the change in COST is in relation to:

$$\Delta C \approx \nabla C \cdot \Delta v.$$

Again, it is the change of the Cost function is in relation to the rate of change of C times the change in v. v of course is our “weights” and “biases” in the neural network.

Suppose we choose: $\Delta v = -\eta \nabla C,$

eta is the LEARNING RATE OF THE NETWORK.

THE GRADIENT DESCENT CONCEPT: ADD IN LEARNING RATE

Based on last slide, we know that:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \| \nabla C \|^2.$$

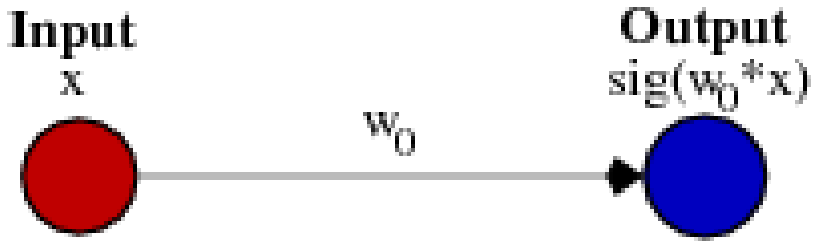
Since the learning rate is always positive, we are always changing in a direction that the COST WILL DECREASE!

So, essentially, we keep changing at the learning rate UNTIL WE REACH GLOBAL MINIMA. How big should the learning rate be then? What happens if it is too small? What happens if it is too big?

HOW IS GRADIENT DESCENT RELATED TO NEURAL NETWORK?

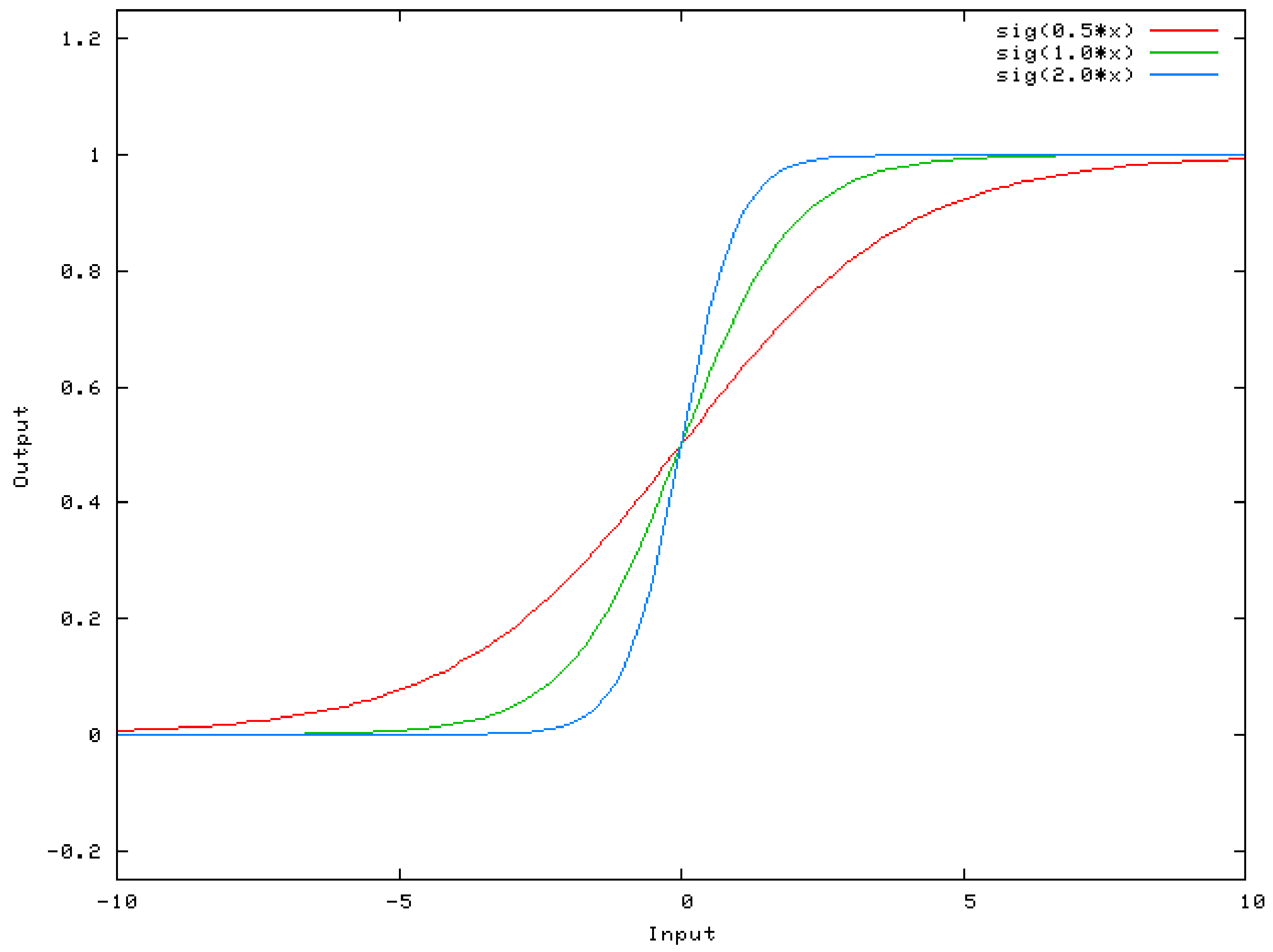
Remember, the whole idea is that we are finding weights and biases to minimize the cost.

Now, for every single neuron in the hidden layer:



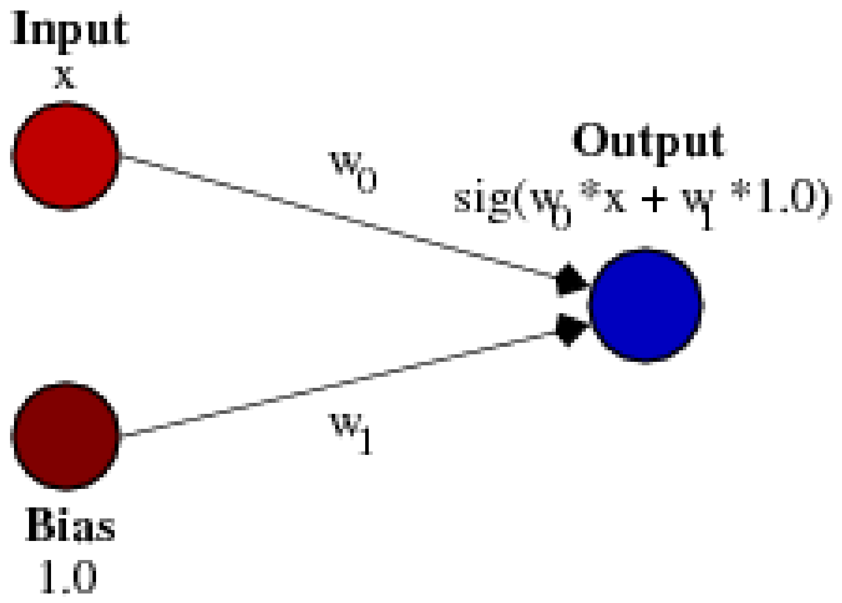
We know that we are simply changing the weights and biases so that when input x goes through the node, the output will be as close to our label as possible. Right?

CHANGING WEIGHTS (3 VALUES) CHANGES THE FORM OF THE FUNCTION

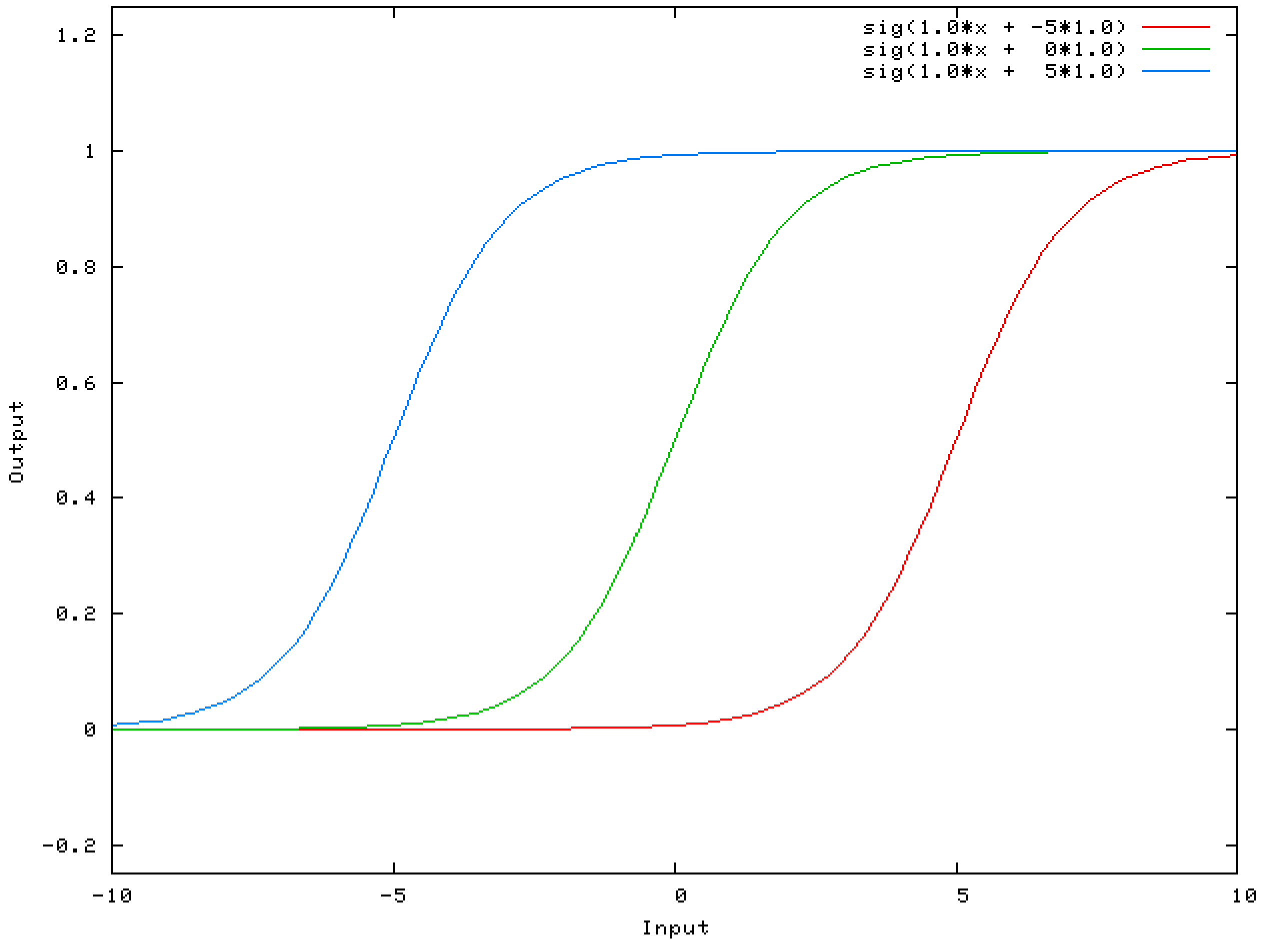


HOW IS GRADIENT DESCENT RELATED TO NEURAL NETWORK? WITH BIAS IN THE PICTURE.

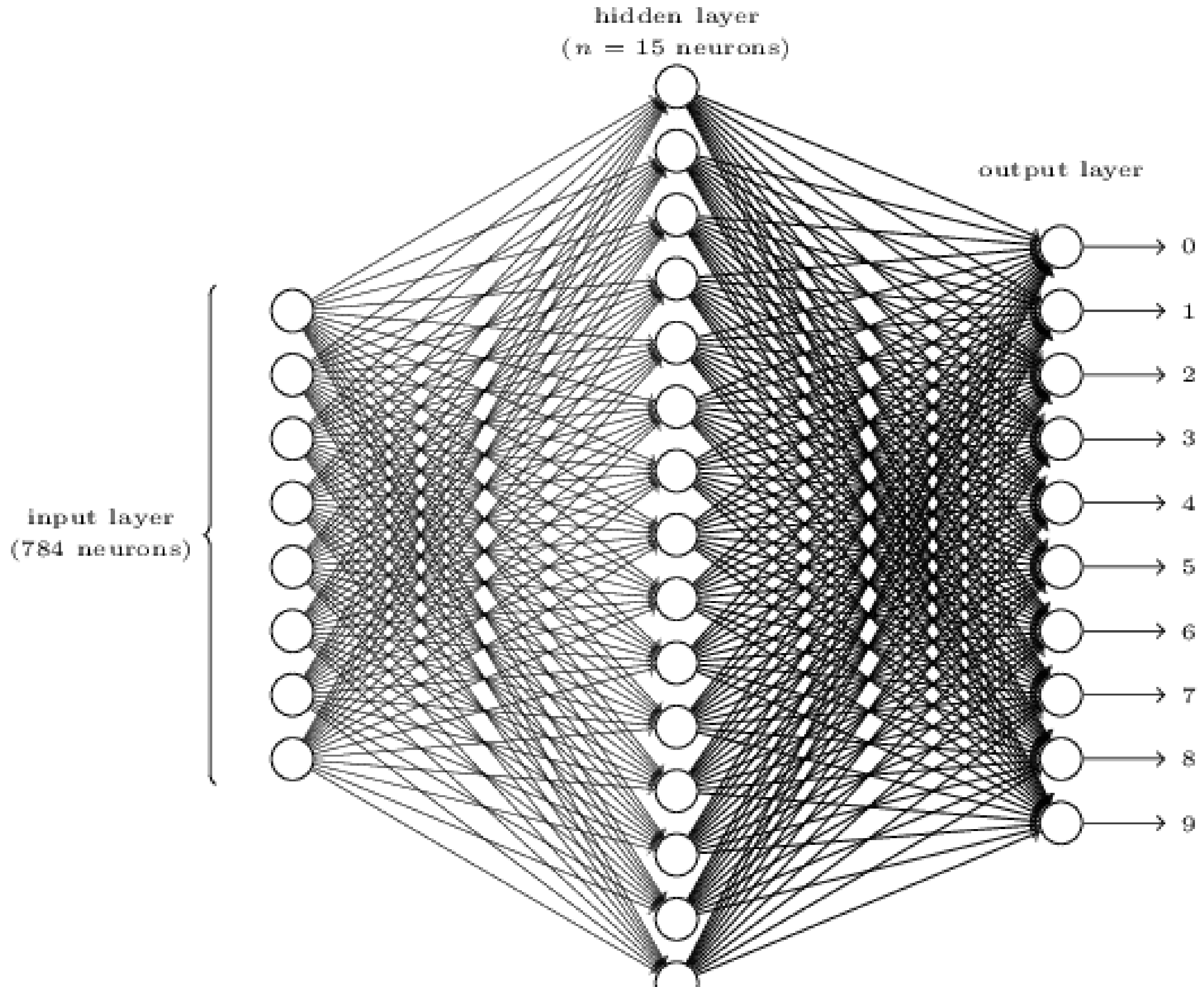
Now, with bias, for every single neuron in the hidden layer:



CHANGING BIASES SHIFTS THE FUNCTION



THE STRUCTURE WE WILL USE



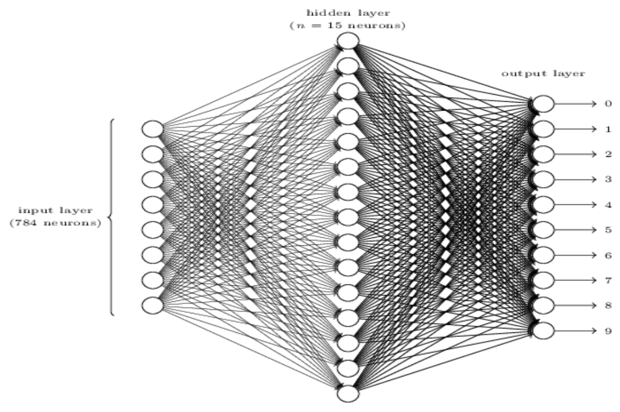
THE NETWORK STRUCTURE

The input pixels are greyscaled. Since you will likely use this structure for SOMETHING ELSE, we will just explain the inputs brieftly.

The value of 1.0 => the pixel is white
The value of 0.9 => the pixel is black

Hidden layer has 15 neurons (a small selection)

Output has 10 neurons. If the first neuron has output of 1, then it means the network is SURE that the input image is a “1”



PYTHON TUTORIALS PART 2

PARAMETERS UNPACKING

More tutorials:

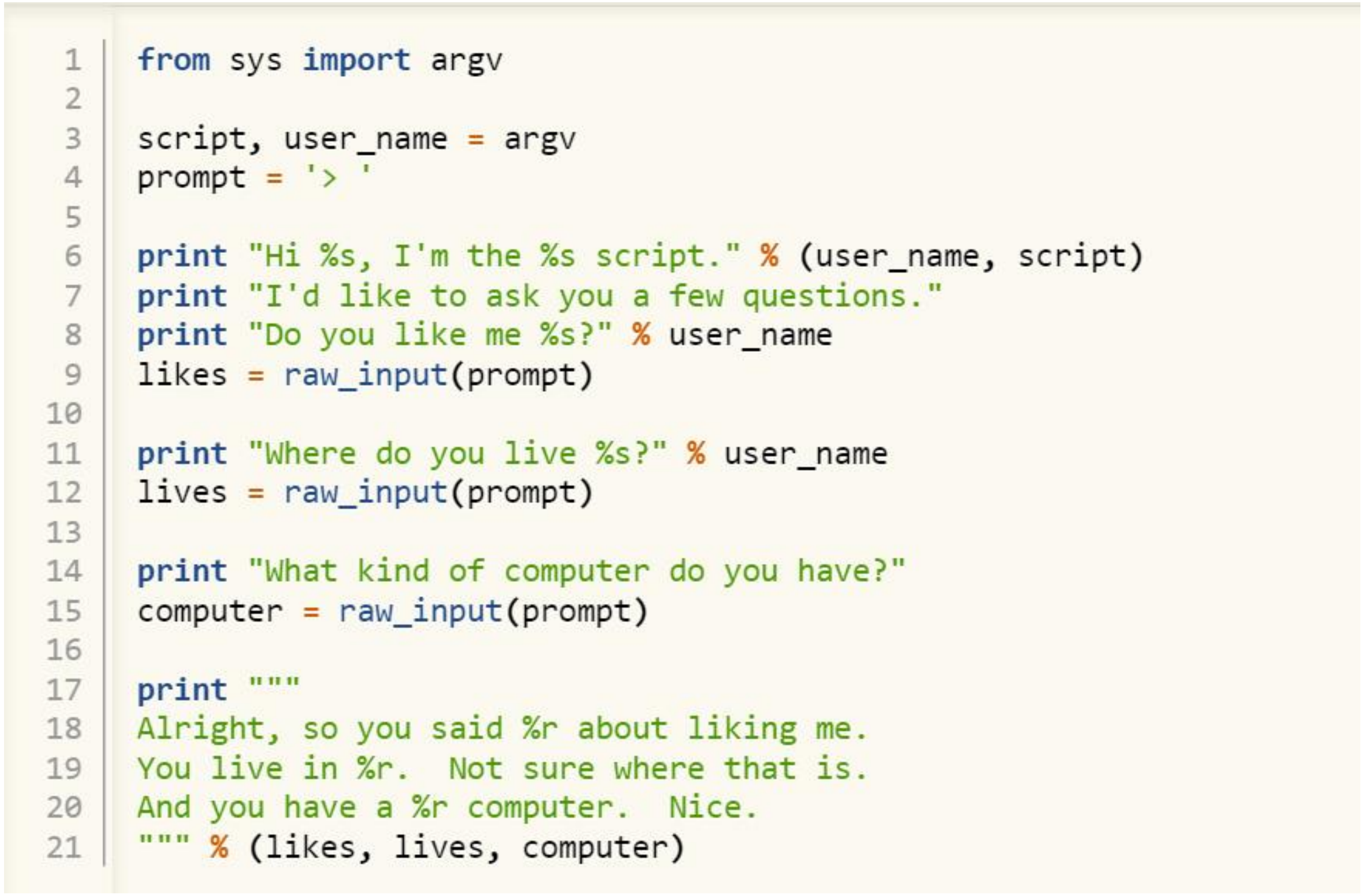
```
1  from sys import argv
2
3  script, first, second, third = argv
4
5  print "The script is called:", script
6  print "Your first variable is:", first
7  print "Your second variable is:", second
8  print "Your third variable is:", third
```

On line 1 we have what's called an "import." This is how you add features to your script from the Python feature set. Rather than give you all the features at once, Python asks you to say what you plan to use. This keeps your programs small, but it also acts as documentation for other programmers who read your code later.

The argv is the "argument variable," a very standard name in programming, that you will find used in many other languages. This variable *holds* the arguments you pass to your Python script when you run it. In the exercises you will get to play with this more and see what happens.

Line 3 "unpacks" argv so that, rather than holding all the arguments, it gets assigned to four variables you can work with: script, first, second, and third. This may look strange, but "unpack" is probably the best word to describe what it does. It just says, "Take whatever is in argv, unpack it, and assign it to all of these variables on the left in order."

PROMPTING AND PASSING

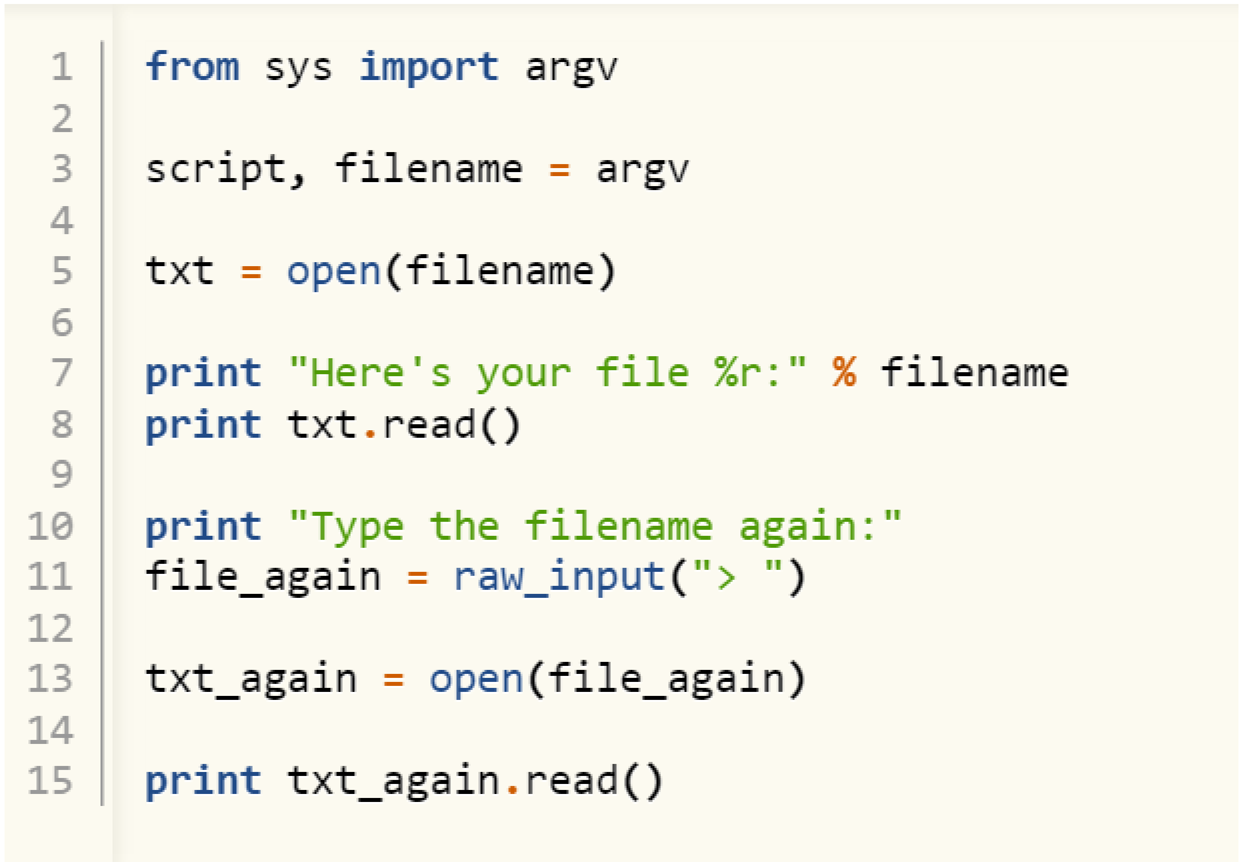


```python
from sys import argv

script, user_name = argv
prompt = '> '

print "Hi %s, I'm the %s script." % (user_name, script)
print "I'd like to ask you a few questions."
print "Do you like me %s?" % user_name
likes = raw_input(prompt)

print "Where do you live %s?" % user_name
lives = raw_input(prompt)

print "What kind of computer do you have?"
computer = raw_input(prompt)

print """
Alright, so you said %r about liking me.
You live in %r.  Not sure where that is.
And you have a %r computer.  Nice.
""" % (likes, lives, computer)
```

READING FILES (TRY NOW)

Save a text file somewhere (in the same folder as your codes),

“this is some random stuff I am typing into the file”

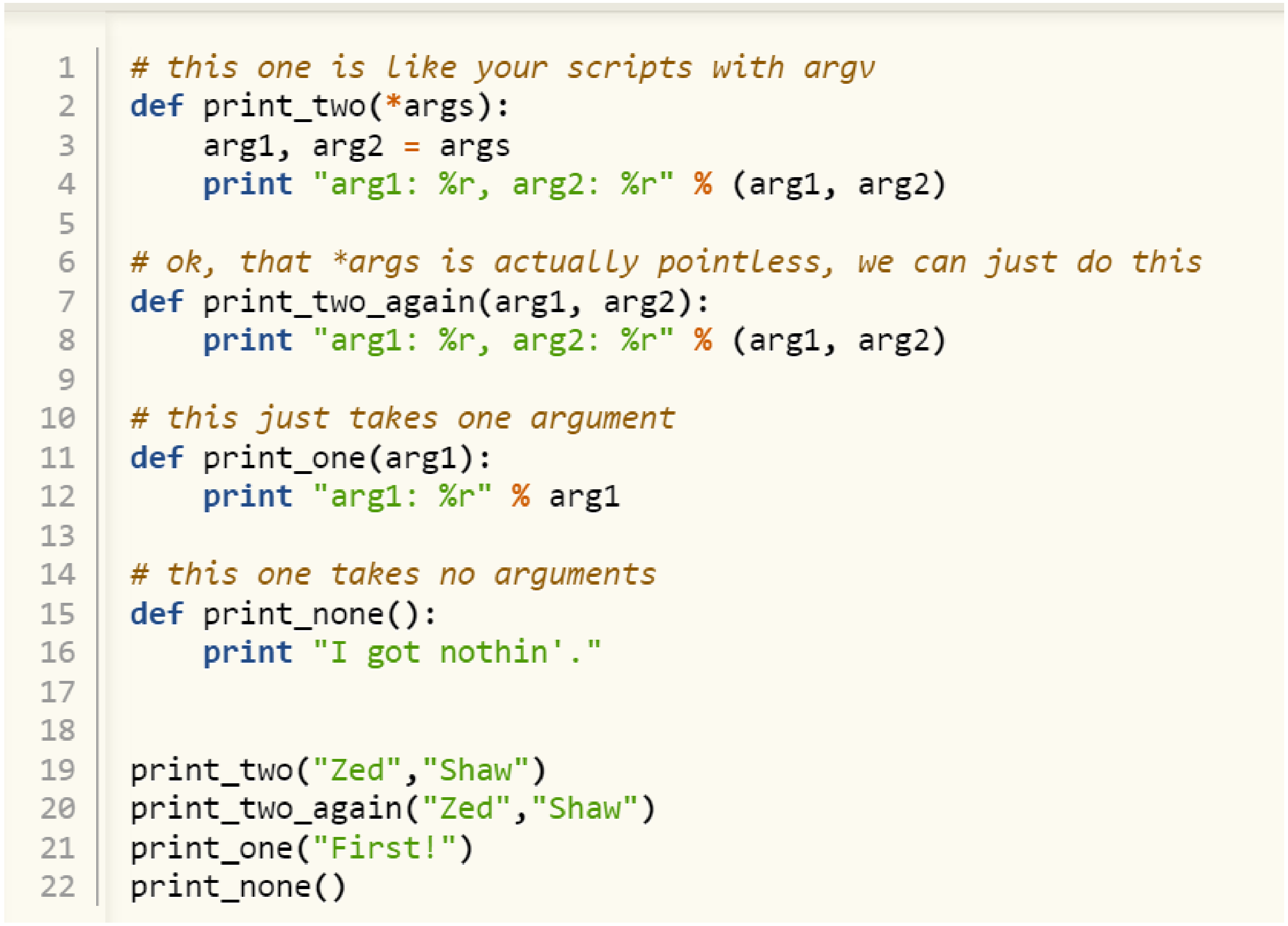


```python
from sys import argv

script, filename = argv

txt = open(filename)

print "Here's your file %r:" % filename
print txt.read()

print "Type the filename again:"
file_again = raw_input("> ")

txt_again = open(file_again)

print txt_again.read()
```

READING AND WRITING FILES

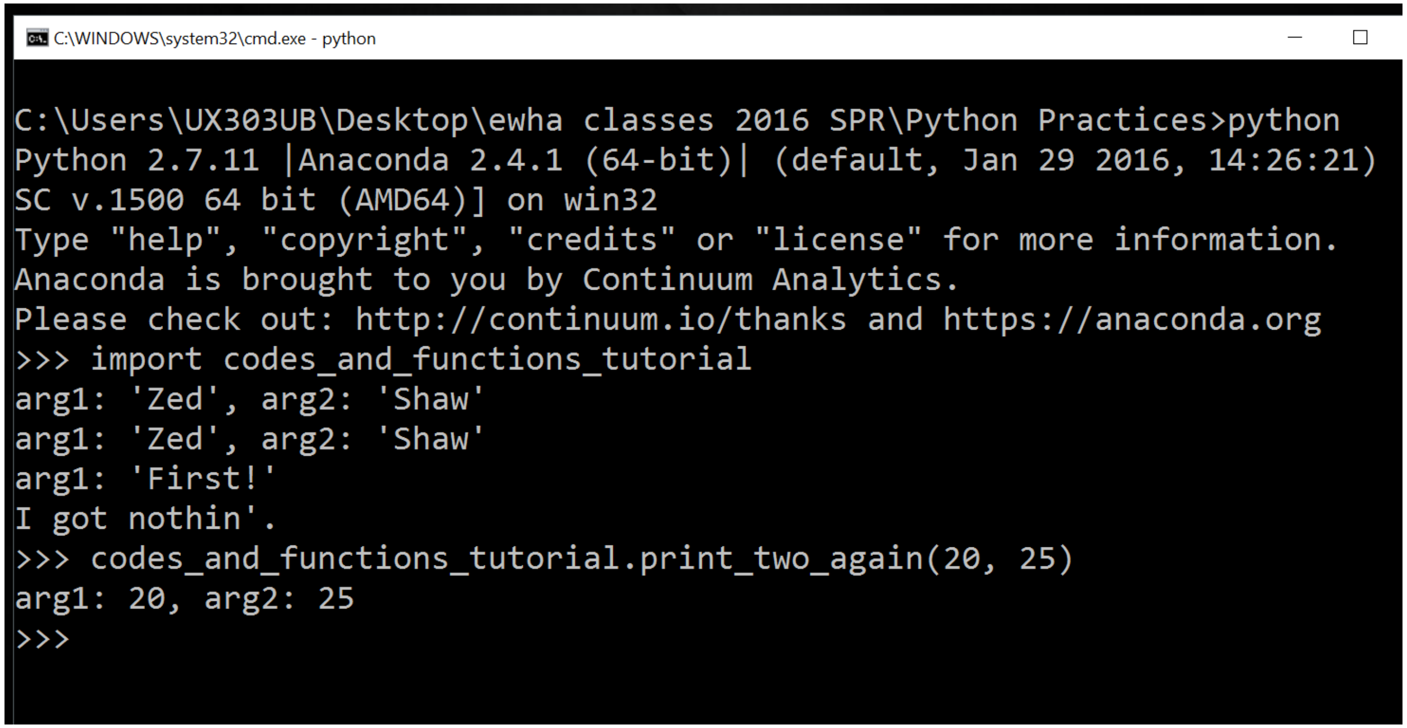
```python
from sys import argv

script, filename = argv

print "We're going to erase %r." % filename
print "If you don't want that, hit CTRL-C (^C)."
print "If you do want that, hit RETURN."

raw_input("?")

print "Opening the file..."
target = open(filename, 'w')

print "Truncating the file.  Goodbye!"
target.truncate()

print "Now I'm going to ask you for three lines."

line1 = raw_input("line 1: ")
line2 = raw_input("line 2: ")
line3 = raw_input("line 3: ")

print "I'm going to write these to the file."

target.write(line1)
target.write("\n")
target.write(line2)
target.write("\n")
target.write(line3)
target.write("\n")

print "And finally, we close it."
target.close()
```

CODES AND CREATING FUNCTIONS!



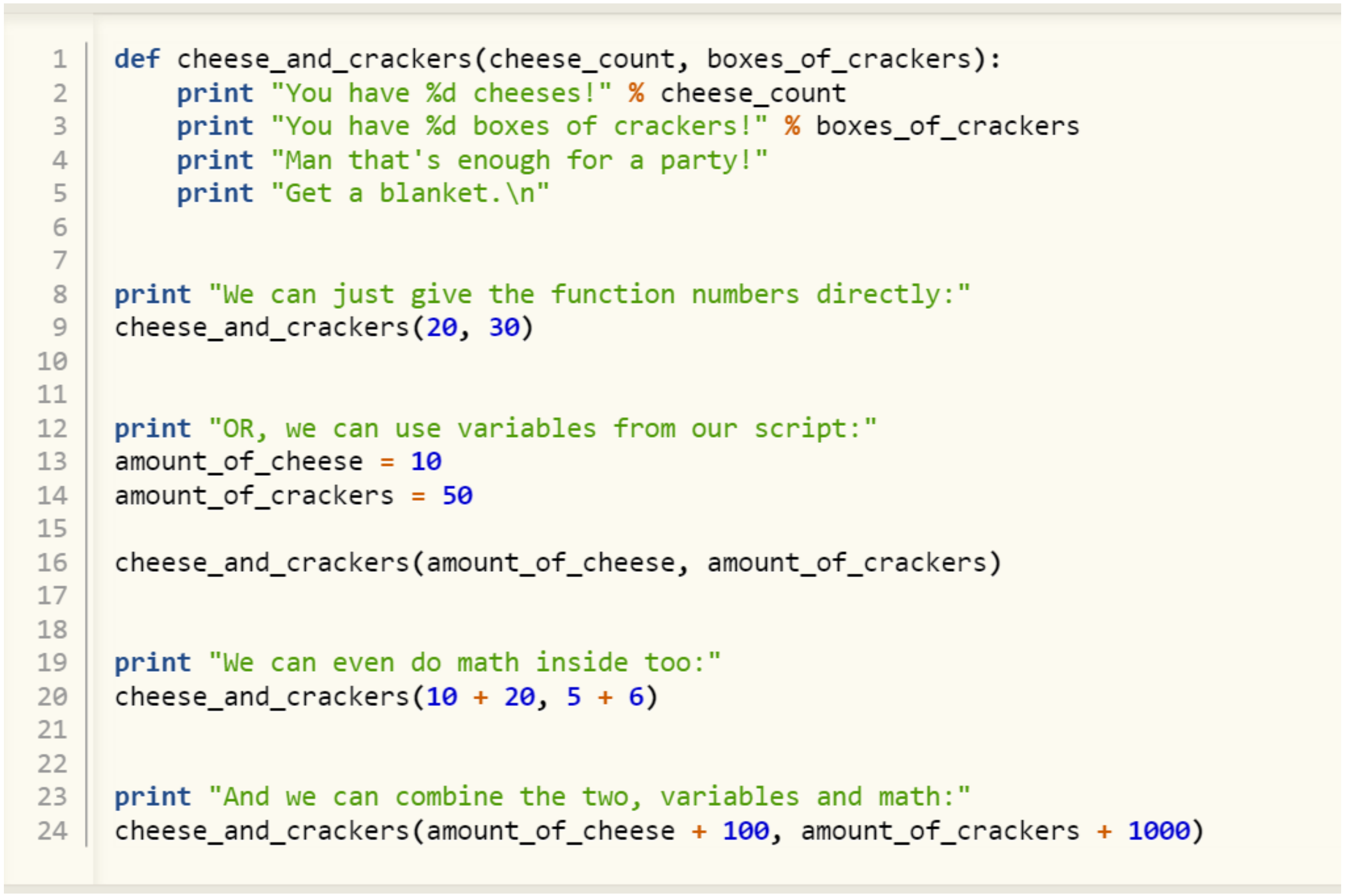
```python
# this one is like your scripts with argv
def print_two(*args):
    arg1, arg2 = args
    print "arg1: %r, arg2: %r" % (arg1, arg2)

# ok, that *args is actually pointless, we can just do this
def print_two_again(arg1, arg2):
    print "arg1: %r, arg2: %r" % (arg1, arg2)

# this just takes one argument
def print_one(arg1):
    print "arg1: %r" % arg1

# this one takes no arguments
def print_none():
    print "I got nothin'."


print_two("Zed","Shaw")
print_two_again("Zed","Shaw")
print_one("First!")
print_none()
```

WHAT YOU SHOULD SEE



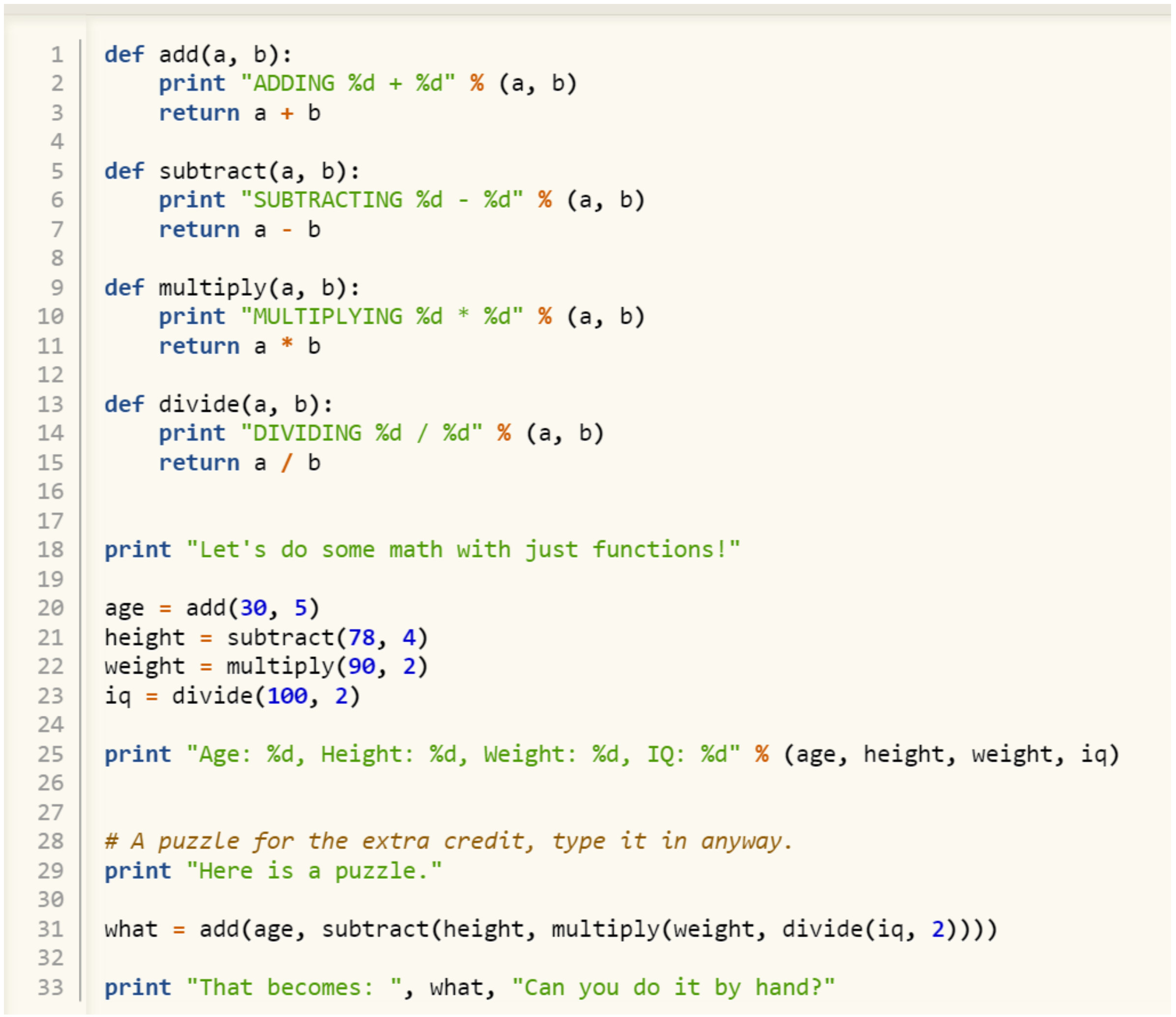
```
C:\Users\UX303UB\Desktop\ewha classes 2016 SPR\Python Practices>python
Python 2.7.11 |Anaconda 2.4.1 (64-bit)| (default, Jan 29 2016, 14:26:21)
SC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>> import codes_and_functions_tutorial
arg1: 'Zed', arg2: 'Shaw'
arg1: 'Zed', arg2: 'Shaw'
arg1: 'First!'
I got nothin'.
>>> codes_and_functions_tutorial.print_two_again(20, 25)
arg1: 20, arg2: 25
>>>
```

CREATING MORE FUNCTIONS!



```python
def cheese_and_crackers(cheese_count, boxes_of_crackers):
    print "You have %d cheeses!" % cheese_count
    print "You have %d boxes of crackers!" % boxes_of_crackers
    print "Man that's enough for a party!"
    print "Get a blanket.\n"


print "We can just give the function numbers directly:"
cheese_and_crackers(20, 30)


print "OR, we can use variables from our script:"
amount_of_cheese = 10
amount_of_crackers = 50

cheese_and_crackers(amount_of_cheese, amount_of_crackers)


print "We can even do math inside too:"
cheese_and_crackers(10 + 20, 5 + 6)


print "And we can combine the two, variables and math:"
cheese_and_crackers(amount_of_cheese + 100, amount_of_crackers + 1000)
```

GET THE FUNCTIONS TO GIVE YOU SOMETHING BACK! (RETURN SOMETHING)



```python
def add(a, b):
    print "ADDING %d + %d" % (a, b)
    return a + b

def subtract(a, b):
    print "SUBTRACTING %d - %d" % (a, b)
    return a - b

def multiply(a, b):
    print "MULTIPLYING %d * %d" % (a, b)
    return a * b

def divide(a, b):
    print "DIVIDING %d / %d" % (a, b)
    return a / b


print "Let's do some math with just functions!"

age = add(30, 5)
height = subtract(78, 4)
weight = multiply(90, 2)
iq = divide(100, 2)

print "Age: %d, Height: %d, Weight: %d, IQ: %d" % (age, height, weight, iq)


# A puzzle for the extra credit, type it in anyway.
print "Here is a puzzle."

what = add(age, subtract(height, multiply(weight, divide(iq, 2))))

print "That becomes: ", what, "Can you do it by hand?"
```